# Java内存区域与内存溢出异常

## 2.2运行时数据区域

## 2.3虚拟机对象探秘

对象的创建

当Java虚拟机遇到一条字节码new指令时， 首先将去检查这个指令的参数是否能在常量池中定位到一个类的符号引用， 并且检查这个符号引用代表的类是否已被加载、 解析和初始化过。 如果没有， 那必须先执行相应的类加载过程，

对象的内存布局

在HotSpot虚拟机里， 对象在堆内存中的存储布局可以划分为三个部分： 对象头（Header） 、 实例数据（Instance Data） 和对齐填充（Padding） 。

对齐填充， 这并不是必然存在的， 也没有特别的含义， 它仅仅起着占位符的作用。 由于HotSpot虚拟机的自动内存管理系统要求对象起始地址必须是8字节的整数倍， 换句话说就是任何对象的大小都必须是8字节的整数倍。 对象头部分已经被精心设计成正好是8字节的倍数（1倍或者2倍） ， 因此， 如果对象实例数据部分没有对齐的话， 就需要通过对齐填充来补全。

对象的访问定位

Java程序会通过栈上的reference数据来操作堆上的具体对象。 由于reference类型在《Java虚拟机规范》 里面只规定了它是一个指向对象的引用， 并没有定义这个引用应该通过什么方式去定位、 访问到堆中对象的具体位置， 所以对象访问方式也是由虚拟机实

现而定的， 主流的访问方式主要有使用 句柄和 直接指针两种方式。

## 2.4OutOfMemoryError异常

出现Java堆内存溢出时， 解决这个内存区域的异常， 第一步首先应确认内存中导致OOM的对象是否是必要的， 也就是要先分清楚到底是出现了内存泄漏（Memory Leak） 还是内存溢出（Memory Overflow） 。如果是内存泄漏， 可进一步通过工具查看泄漏对象到GC Roots的引用链， 找到泄漏对象是通过怎样的引用路径、 与哪些GC Roots相关联， 才导致垃圾收集器无法回收它们， 根据泄漏对象的类型信息以及它到GC Roots引用链的信息， 一般可以比较准确地定位到这些对象创建的位置， 进而找出产生内存泄漏的代码的具体位置。